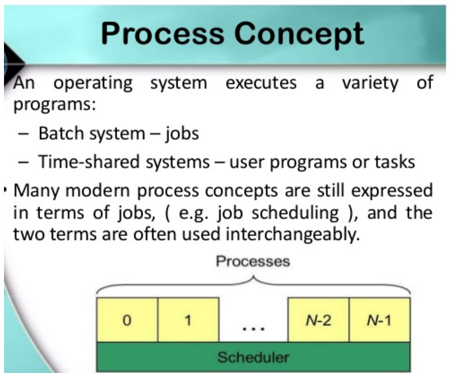
Date=02/09/2020
Lecture By=Arkesh Jaiswal
Subject ⇒ Operating System-2

| IN PREVIOUS LECTURE **(QUICK RECAP) Date-01/02/2020** | **In Today's Lecture (Overview)** |
|---|---|
| What is operating software?<br><br>Why Programming Directly On Hardware Is Not possible?<br><br>Functions of Operating System<br><br>What is kernel In operating system<br><br>Functions of a Kernel<br><br>What Is System Calls<br><br>Mcqs<br><br>Questions for Self Practice | ⇒ What is Process<br><br>⇒ Elements of Process<br><br>⇒ MCQs<br><br>⇒ Questions for self practice / Assignment for the Day |

## ⇒ What is Process

=A process is an instance of a computer program that is being executed.

=It contains the program code and its activity. Depending on the operating system (OS)
=a **process** may be made up of multiple threads of execution that execute instructions concurrently.

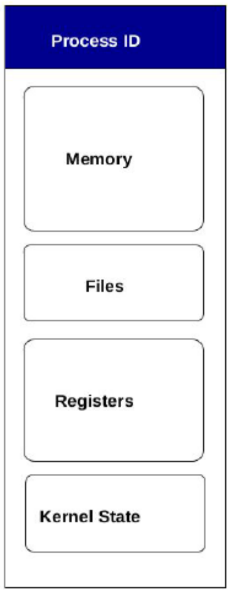


### Process Concept

- An operating system executes a variety of programs:
  - Batch system – jobs
  - Time-shared systems – user programs or tasks
- Many modern process concepts are still expressed in terms of jobs, ( e.g. job scheduling ), and the two terms are often used interchangeably.

Processes
0
1
...
N-2
N-1
Scheduler

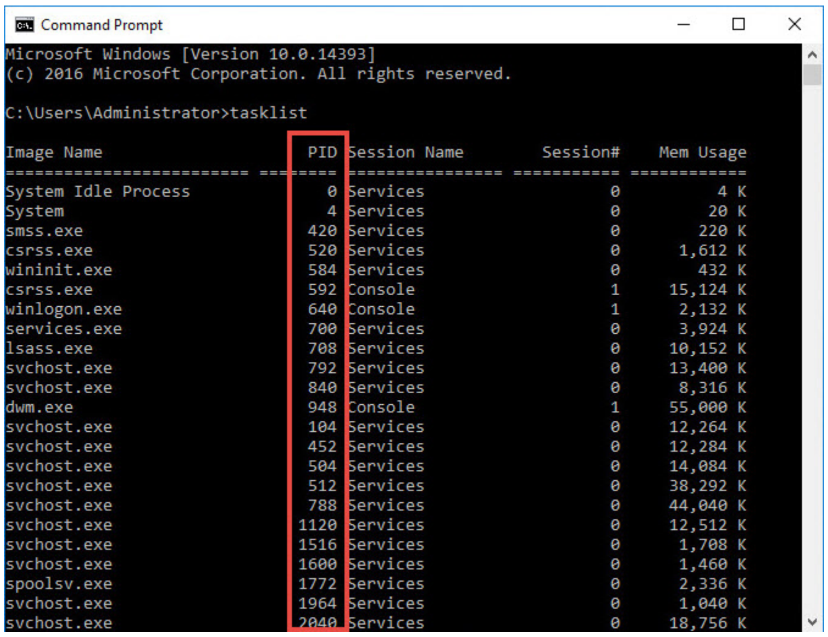
## ⇒ Elements of Process



Shown below are the Elements of the Process

**==> Process ID**

The process ID (or the PID) is assigned by the operating system and is unique to each running process.



```
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>tasklist

Image Name                     PID Session Name        Session#    Mem Usage
========================= ======== ================ =========== ============
System Idle Process              0 Services                   0          4 K
System                           4 Services                   0         20 K
smss.exe                       420 Services                   0        220 K
csrss.exe                      520 Services                   0      1,612 K
wininit.exe                    584 Services                   0        432 K
csrss.exe                      592 Console                    1     15,124 K
winlogon.exe                   640 Console                    1      2,132 K
services.exe                   700 Services                   0      3,924 K
lsass.exe                      708 Services                   0     10,152 K
svchost.exe                    792 Services                   0     13,400 K
svchost.exe                    840 Services                   0      8,316 K
dwm.exe                        948 Console                    1     55,000 K
svchost.exe                    104 Services                   0     12,264 K
svchost.exe                    452 Services                   0     12,284 K
svchost.exe                    504 Services                   0     14,084 K
svchost.exe                    512 Services                   0     38,292 K
svchost.exe                    788 Services                   0     44,040 K
svchost.exe                   1120 Services                   0     12,512 K
svchost.exe                   1516 Services                   0      1,708 K
svchost.exe                   1600 Services                   0      1,460 K
spoolsv.exe                   1772 Services                   0      2,336 K
svchost.exe                   1964 Services                   0      1,040 K
svchost.exe                   2040 Services                   0     18,756 K
```

**==> Memory**

We will learn exactly how a process gets it's memory in the following weeks -- it is one of the most fundamental parts of how the operating system works. However, for now it is sufficient to know that each process gets it's own section of memory.

In this memory all the program code is stored, along with variables and any other allocated storage.

Parts of the memory can be shared between processes (called, not surprisingly *shared memory*). You will often see this called *System Five Shared Memory* (or SysV SHM) after the original implementation in an older operating system.

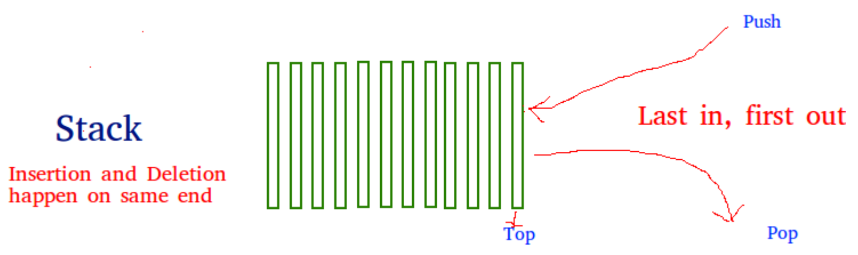
**==> Code and Data**

A process can be further divided into *code* and data sections. Program code and data should be kept separately since they require different permissions from the operating system and separation facilitates sharing of code (as you see later). The operating system needs to give program code permission to be read and executed, but generally not written to. On the other hand data (variables) require read and write permissions but should not be executable.

**==> The Stack**

One other very important part of a process is an area of memory called *the stack*. This can be considered part of the data section of a process, and is intimately involved in the execution of any program.

A stack is generic data structure that works exactly like a stack of plates; you can *push* an item (put a plate on top of a stack of plates), which then becomes the top item, or you can *pop* an item (take a plate off, exposing the previous plate).



**==> The Heap**

The heap is an area of memory that is managed by the process for on the fly memory allocation. This is for variables whose memory requirements are not known at compile time.

The bottom of the heap is known as the *brk*, so called for the system call which modifies it. By using the brk call to grow the area downwards the process can request the kernel allocate more memory for it to use.

The heap is most commonly managed by the malloc library call. This makes managing the heap easy for the programmer by allowing them to simply allocate and free (via the free call) heap memory.

**==> Memory Layout**

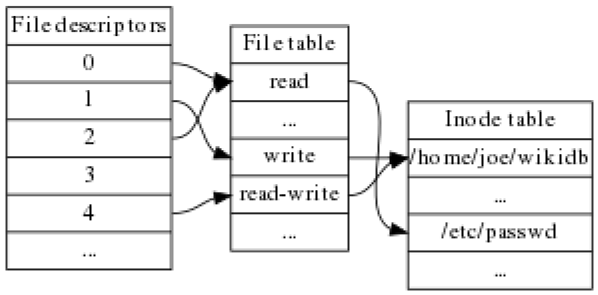
As we have seen a process has smaller areas of memory allocated to it, each with a specific purpose.

An example of how the process is laid out in memory by the kernel is given above. Starting from the top, the kernel reserves its self some memory at the top of the process (we see with virtual memory how this memory is actually shared between all processes).

**==> File Descriptors**

In the first week we learnt about stdin, stdout and stderr; the default files given to each process. You will remember that these files always have the same file descriptor number (0,1,2 respectively).

Thus, file descriptors are kept by the kernel individually for each process.



**==> Registers**

We know that the processor essentially performs generally simple operations on values in registers. These values are read (and written) to memory -- we mentioned above that each process is allocated memory which the kernel keeps track of.

**==> Kernel State**

Internally, the kernel needs to keep track of a number of elements for each process.

**==> Process State**

Another important element for the operating system to keep track of is the process state. If the process is currently running it makes sense to have it in a *running* state.

**==> Priority**

Some processes are more important than others, and get a higher priority. See the discussion on the scheduler below.

**==> Statistics**

The kernel can keep statistics on each process's behaviour which can help it make decisions about how the process behaves; for example does it mostly read from disk or does it mostly do CPU intensive operations?

## ⇒ MCQs

1.**Which of these is not a part of a process?**

A=stack

B=heap

C=queue

D=data

2.**Which statement is true?**

A=A process is a segment of a thread

B=A thread is a segment of a process

3.**All mathematical operations of a process happens through?**

A=registers

B=ledgers

C=bytes

## ⇒ Questions for self practice / Assignment for the Day

==> Elaborate and explain with a diagram difference between processes and threads.